# Semantic Segmentation on VOC 2007 Dataset

**Madeleine C. Kerr**
Scripps Institution of Oceanography
University of California, San Diego
La Jolla, CA 91211
mkerr@ucsd.edu

**Cameron Cinel**
Department of Mathematics
University of California, San Diego
La Jolla, CA 91211
ccinel@ucsd.edu

**Rishabh Bhattacharya**
Department of Mechanical & Aerospace Engineering
University of California, San Diego
La Jolla, CA 91211
ribhattacharya@ucsd.edu

**Shrey Kansal**
Department of Mechanical & Aerospace Engineering
University of California, San Diego
La Jolla, CA 91211
skansal@ucsd.edu

## Abstract

The project explores semantic segmentation as a computer vision technique used to label pixels in an image based on the obejcts they belong to with various applications such as in autonomous driving and medical imaging. Deep learning models, such as convolutional neural networks, are commonly used for semantic segmentation but face issues like slow convergence and high computational overhead. The project aims to improve the baseline model's performance by exploring techniques like batch normalization, Xavier initialization, cosine annealing learning rate, input image transformations, and using a weighted loss function to address the class imbalance problem. We trained four different models on the PASCAL VOC 2007 dataset: two custom models, ResNet34, and UNet. We achieved our best performance with ResNet34 with a .1288 mean IoU and a pixel accuracy of 24.37%. In addition, we tested various other methods to improve performance, such as data augmentation, different loss functions, and adaptive learning rates.

## 1 Introduction

Semantic segmentation is a computer vision technique that labels each pixel of an image based on the object it belongs to, and has various applications such as autonomous driving and medical imaging. Deep learning models, particularly convolutional neural networks, are commonly used for semantic segmentation and are trained on large datasets. However, such models face challenges such as long convergence rate, high computational overhead, and class imbalances, among others.

This report explores different techniques and networks to improve the performance of the baseline semantic segmentation model. The techniques include batch normalization [3], Xavier initialization [9], cosine annealing learning rate [6], and image transformations such as rotation and flipping. The networks investigated include UNet [8], FCN8 [5] and ResNet [2], which leverage skip connections to improve gradient propagation and feature learning.

Additionally, a weighted loss function is used to address the class imbalance problem, where there are typically more background pixels than object pixels in image segmentation problems. Our experiments show that these techniques and networks improve the performance of the baseline model, with ResNet achieving the best results.

In conclusion, this report provides insights on how to improve the performance of semantic segmentation models using various techniques and networks. The results demonstrate the effectiveness of these approaches in addressing common challenges in semantic segmentation.

## 2 Related Work

Recent advancements in semantic segmentation have been focused on improving the accuracy and efficiency of deep learning models. Fully Convolutional Networks (FCN) [5] and U-Net [8] are two popular architectures for semantic segmentation. FCN utilizes a series of convolutional layers to learn the feature representations of the input image, which are then upsampled using transpose convolutional layers to obtain the final segmentation map. U-Net, on the other hand, uses skip connections to retain high-level information during the upsampling process, enabling better recovery of fine-grained details in the segmentation map. Both FCN and U-Net have been widely used in various applications, including medical imaging and autonomous driving.

Another recent development in semantic segmentation is DeepLabV3+ [1], which combines the depth-wise separable convolution operation and atrous convolution operation to learn multi-scale contextual information. DeepLabV3+ also utilizes the image pyramid network to improve the accuracy of the segmentation results across different scales.

Focal Loss [4] is another technique that has been recently introduced to improve the performance of deep learning models for semantic segmentation. Focal loss is designed to address the class imbalance problem, which occurs when the number of pixels belonging to one class heavily outweighs the other classes in the training dataset. This allows the model to better distinguish between different classes and improve the accuracy of the segmentation results.

## 3 Methods

### 3.1 Baseline

Our initial model was a simple fully convolutional neural network consisting of a 5 layer encoder, a 5 layer decoder, and a single ouput convolutional layer. Our architecture is described in Table 1. Additionally, between each layer we applied batch-norm to the outputs of the previous layer. Our loss function for our network is cross entropy with no class weighting:

$$\mathcal{L} = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_k^{(n)} \ln(y_k^{(n)})$$

where $y_k^{(n)}$ is the predicted probability of the pixel $y$ of the $n$-th example belonging to the $k$-th class and $t_k^{(n)}$ is 1 if the pixel belongs to the $k$-th class in the ground truth and 0 otherwise.

We initialized our model's weights using Xavier initialization. The hyperparameters we used for training are the following: learning rate of 0.005, 100 maximum training epochs, batch size of 16, and weight decay (L2 penalty) of 0.01. Additionally, we shuffled our training data in each epoch of training. We used the Adam optimizer and implementing early stopping, ending training after 10 epochs without reduction in our validation set's loss.

### 3.2 Improvements over Baseline

In order to improved our results using the same baseline architecture, we modified our training algorithm in a few ways. First, we added a cosine annealing learning rate scheduler. This modified our learning rate at epoch $t$ via

$$\eta_t = \frac{\eta_0}{2}\left(1 + \cos\left(\frac{\pi t}{t_{max}}\right)\right)$$

| Section | Layer | In-Channels | Out-Channels | Kernel | Padding | Stride | Activation |
|---|---|---|---|---|---|---|---|
| Encoder | Convolution 1 | 3 | 32 | 3 | 2 | 1 | ReLU |
| | Convolution 2 | 32 | 64 | 3 | 2 | 1 | ReLU |
| | Convolution 3 | 64 | 128 | 3 | 2 | 1 | ReLU |
| | Convolution 4 | 128 | 256 | 3 | 2 | 1 | ReLU |
| | Convolution 5 | 256 | 512 | 3 | 2 | 1 | ReLU |
| Decoder | Deconvolution 1 | 512 | 512 | 3 | 2 | 1 | ReLU |
| | Deconvolution 2 | 512 | 256 | 3 | 2 | 1 | ReLU |
| | Deconvolution 3 | 256 | 128 | 3 | 2 | 1 | ReLU |
| | Deconvolution 4 | 128 | 64 | 3 | 2 | 1 | ReLU |
| | Deconvolution 5 | 64 | 32 | 3 | 2 | 1 | ReLU |
| Classifier | Final Convolution | 32 | 21 | 1 | 0 | 1 | Softmax |

Table 1: Architecture for our baseline model

where $\eta_0$ is our initial learning rate and $t_{max}$ is the maximum possible number of epochs, in our case 100. This allowed us to train at a high learning rate initially, which would then taper down in order to better converge to the minima.

Additionally, we augmented our dataset through random transformations. We randomly flip our images, both horizontally and vertically, randomly rotated them, and randomly cropped them. Since our targets were also images, we applied identical transformations to our target masks in order for our model to learn properly. This would help give robustness to our model, as it would build in some rotational and symmetry invariance.

Finally, we changed our loss function in order to account for our class imbalance. Since the majority of the pixels of each image belonged to the background class and since every image only contained a few non-background classes, the overwhelming majority of our total pixels belonged to the background class. In order to prevent our model from simply predicting everything to be the background, we implemented a few different loss functions.

The first was weighted cross entropy. For this, we calculated class weights $w_k$ for each of our classes $k$ by simply taking the multiplicative inverse of their class frequency in the training dataset, i.e. the total number of pixels in the dataset belonging to the class $k$. We then modified our loss function to be

$$\mathcal{L} = -\sum_{n=1}^{N}\sum_{k=1}^{K} w_k t_k^{(n)} \ln(y_k^{(n)})$$

Since rarer classes have higher weights, this encourages our model to learn the rarer classes, rather than just learning the background.

We additionally tried to use Dice Loss [7] which has been used in image segmentation for medical imaging. This loss function is much more closely related to the IoU metric we are using to evaluate our models, and thus our model would better optimize this IoU metric. The loss is calculated as

$$\mathcal{L} = 1 - 2\frac{\sum_{n=1}^{N}\sum_{k=1}^{K} t_k^{(n)} y_k^{(n)} + \varepsilon}{\sum_{n=1}^{N}\sum_{k=1}^{K} (y_k^{(n)} + t_k^{(n)}) + \varepsilon}$$

where $\varepsilon > 0$ is a smoothing factor. The fraction on the right in the loss function is an analytic way of calculating the intersection of the prediction and ground truth, divided by the the sum of the areas of prediction and ground truth for each of the $K$ classes. Additionally, we calculated dice losses individually for each class and weighted them similar to cross entropy.

The final loss we tried was Focal Loss [4]. For a given pixel output pixel $y$ and target pixel $t$, we can view the cross entropy loss as

$$CE(y, t) = CE(y_t) = -\ln(y_t)$$

We then calculate the focal loss as

$$\mathcal{L} = -(1 - y_t)^{\gamma} \ln(y_t)$$

where $\gamma > 0$ is called the focal power. Essentially, focal loss prioritizes the model to stop focusing on well-classified examples and move onto learning more from more difficult examples. Additionally, we weighted our focal loss similar to our cross entropy loss.

### 3.3 Experimentation

#### 3.3.1 Custom Architecture

We implemented a custom architecture taking inspiration from the famous FCN8 [**?**] and VGG16 [**?**] decoder. The difference between the baseline and our custom architecture is with respect to the number of layers and parameters, inclusion of maxpool layers and the addition of skip connections. A number of different permutations were tried by rearranging the convolution and maxpool layers. The model was tested with varying kernel sizes, dilation and padding as well. The learning rate and weight decay (regularization constant) were chosen accordingly.

For the encoder part of the network, we implemented seven convolution blocks, each followed by a maxpool layer. After the first convolution and maxpool layer, we implemented four blocks, each comprising of 2-3 convolution layers and a maxpool layer. Batch normalization was implemented after applying non-linearity to each convolution layer. At the end, the last convolution layer was reduced from $4096$ channels to $n\_class$ channels in order to obtain to project each feature in a separate channel.

For the decoder part of the network, we used skip connections to incorporate the information from corresponding encoder layer by concatenating it and then deconvoluting it to upsample the feature map. In total, four deconvolution layers were implemented to upsample the feature map to original image size with $n\_class$ channels. Maxpool and ReLU were applied as well, to maintain translational invariance and non-linearity in the features.

The precise details pertaining to the layer sizes can be referred to from Table 2.

| Section | Layer | In-Channels | Out-Channels | Kernel | Padding | Stride | Activation |
|---|---|---|---|---|---|---|---|
| Encoder | Convolution 1 | 3 | 64 | 3 | 1 | 1 | ReLU |
| | Convolution 2 | 64 | 64 | 3 | 1 | 1 | ReLU |
| | MaxPool 1 | 64 | 64 | 2 | 2 | 1 | - |
| | Convolution 3 | 64 | 128 | 3 | 1 | 1 | ReLU |
| | Convolution 4 | 128 | 256 | 3 | 1 | 1 | ReLU |
| | MaxPool 2 | 128 | 128 | 2 | 2 | 1 | - |
| | Convolution 5 | 256 | 256 | 3 | 1 | 1 | ReLU |
| | Convolution 6 | 256 | 256 | 3 | 1 | 1 | ReLU |
| | MaxPool 3 | 256 | 256 | 2 | 2 | 1 | - |
| | Convolution 7 | 256 | 512 | 3 | 1 | 1 | ReLU |
| | Convolution 8 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | Convolution 9 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | MaxPool 4 | 512 | 512 | 2 | 2 | 1 | - |
| | Convolution 10 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | Convolution 11 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | Convolution 12 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | MaxPool 5 | 512 | 512 | 2 | 2 | 1 | - |
| | Convolution 13 | 512 | 4096 | 3 | 1 | 1 | ReLU |
| | Convolution 14 | 4096 | 4096 | 3 | 1 | 1 | ReLU |
| | Convolution 15 | 4096 | 21 | 3 | 1 | 1 | ReLU |
| Decoder | Deconvolution 1 | 21 | 21 | 2 | 2 | 1 | ReLU |
| | Deconvolution 2 | 42 | 21 | 2 | 2 | 1 | ReLU |
| | Deconvolution 3 | 42 | 21 | 2 | 2 | 1 | ReLU |
| | Deconvolution 4 | 42 | 21 | 2 | 2 | 1 | ReLU |
| | Deconvolution 5 | 42 | 21 | 2 | 2 | 1 | ReLU |
| Classifier | Final Convolution | 21 | 21 | 2 | 2 | 1 | Softmax |

Table 2: Architecture for our custom model

#### 3.3.2 ResNet

We experimented with transfer learning using the architecture of Resnet34 [2], a residual network. The specific architecture can be seen in figure 3.

| Section | Layer | In-Channels | Out-Channels | Kernel | Padding | Stride | Activation |
|---|---|---|---|---|---|---|---|
| Encoder | Convolution 1 | 3 | 64 | 7 | 0 | 2 | ReLU |
| | Max Pool 1 | 64 | 64 | 2 | 0 | 2 | - |
| | Convolution 2 | 64 | 64 | 3 | 0 | 1 | ReLU |
| | Convolution 3 | 64 | 64 | 3 | 0 | 1 | ReLU |
| | Convolution 4 | 64 | 64 | 3 | 0 | 1 | ReLU |
| | Convolution 5 | 64 | 64 | 3 | 0 | 1 | ReLU |
| | Convolution 6 | 64 | 64 | 3 | 0 | 1 | ReLU |
| | Convolution 7 | 64 | 64 | 3 | 0 | 1 | ReLU |
| | Convolution 8 | 64 | 128 | 3 | 0 | 2 | ReLU |
| | Convolution 9 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 10 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 11 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 12 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 13 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 14 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 15 | 128 | 128 | 3 | 0 | 1 | ReLU |
| | Convolution 16 | 128 | 256 | 3 | 0 | 2 | ReLU |
| | Convolution 17 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 18 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 19 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 20 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 21 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 22 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 23 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 24 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 25 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 26 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 27 | 256 | 256 | 3 | 0 | 1 | ReLU |
| | Convolution 28 | 256 | 512 | 3 | 0 | 2 | ReLU |
| | Convolution 29 | 512 | 512 | 3 | 0 | 1 | ReLU |
| | Convolution 30 | 512 | 512 | 3 | 0 | 1 | ReLU |
| | Convolution 31 | 512 | 512 | 3 | 0 | 1 | ReLU |
| | Convolution 32 | 512 | 512 | 3 | 0 | 1 | ReLU |
| | Convolution 33 | 512 | 512 | 3 | 0 | 1 | ReLU |
| Decoder | Deconvolution 1 | 512 | 512 | 3 | 2 | 1 | ReLU |
| | Deconvolution 2 | 512 | 256 | 3 | 2 | 1 | ReLU |
| | Deconvolution 3 | 256 | 128 | 3 | 2 | 1 | ReLU |
| | Deconvolution 4 | 128 | 64 | 3 | 2 | 1 | ReLU |
| | Deconvolution 5 | 64 | 32 | 3 | 2 | 1 | ReLU |
| Classifier | Final Convolution | 32 | 21 | 1 | 0 | 1 | Softmax |

Table 3: Architecture for our ResNet34 model

Residual networks are deep networks that are designed to allow for the effective operation of a similarly accurate model that is shallower. It does this using residual connections between certain input and outputs that are several layers apart. This ends up creating a series of "shortcut" connections between different layers which stop the degradation of training accuracy that would occur in similarly deep models without these residual connections. We used the image flipping regularizarion technique where inputs are flipped horizontally and vertically with a probability of 50% along with randomly rotating the images. We trained on batch sizes of 16 and with shuffling each epoch. The validation sets were the same size. We initialized the learning rate to 0.005. We used the Adam optimizer, cosine annealing to adapt the learning rate in each epoch, and our loss function was weighted cross entropy. We used the ReLU activation function and batch normalization between each deconvolutional layer. We address the class imbalance problem using the cross entropy loss function by adding the class weights to the calculation using the PyTorch functionality.

#### 3.3.3 UNet

We implemented UNet as described in [8]. The one difference between our model and the one describe in [8] is that we added padding of 1 to each of the convolutions as to preserve image size through each of the convolutional layers. All other layers and activation functions remain the same. The specific architecture can be seen in 4.

| Section | Layer | In-Channels | Out-Channels | Kernel | Padding | Stride | Activation |
|---|---|---|---|---|---|---|---|
| Encoder | Convolution 1 | 3 | 64 | 3 | 1 | 1 | ReLU |
| | Convolution 2 | 64 | 64 | 3 | 1 | 1 | ReLU |
| | Max Pool 1 | 64 | 64 | 2 | 0 | 2 | - |
| | Convolution 3 | 64 | 128 | 3 | 1 | 1 | ReLU |
| | Convolution 4 | 128 | 128 | 3 | 1 | 1 | ReLU |
| | Max Pool 2 | 128 | 128 | 2 | 0 | 2 | - |
| | Convolution 5 | 128 | 256 | 3 | 1 | 1 | ReLU |
| | Convolution 6 | 256 | 256 | 3 | 1 | 1 | ReLU |
| | Max Pool 3 | 256 | 256 | 2 | 0 | 2 | - |
| | Convolution 7 | 256 | 512 | 3 | 1 | 1 | ReLU |
| | Convolution 8 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | Max Pool 4 | 512 | 512 | 2 | 0 | 2 | - |
| | Convolution 9 | 512 | 1024 | 3 | 1 | 1 | ReLU |
| | Convolution 10 | 1024 | 1024 | 3 | 1 | 1 | ReLU |
| Decoder | Deconvolution 1 | 1024 | 512 | 2 | 0 | 2 | - |
| | Convolution 11 | 1024 | 512 | 3 | 1 | 1 | ReLU |
| | Convolution 12 | 512 | 512 | 3 | 1 | 1 | ReLU |
| | Deconvolution 2 | 512 | 256 | 2 | 0 | 2 | - |
| | Convolution 13 | 512 | 256 | 3 | 1 | 1 | ReLU |
| | Convolution 14 | 256 | 256 | 3 | 1 | 1 | ReLU |
| | Deconvolution 3 | 256 | 128 | 2 | 0 | 2 | - |
| | Convolution 15 | 256 | 128 | 3 | 1 | 1 | ReLU |
| | Convolution 16 | 128 | 128 | 3 | 1 | 1 | ReLU |
| | Deconvolution 4 | 128 | 64 | 2 | 0 | 2 | - |
| | Convolution 17 | 128 | 64 | 3 | 1 | 1 | ReLU |
| | Convolution 18 | 64 | 64 | 3 | 1 | 1 | ReLU |
| | Deconvolution 5 | 64 | 32 | 2 | 0 | 2 | - |
| | Convolution 19 | 64 | 32 | 3 | 1 | 1 | ReLU |
| | Convolution 20 | 32 | 32 | 3 | 1 | 1 | ReLU |
| Classifier | Final Convolution | 32 | 21 | 1 | 0 | 1 | Softmax |

Table 4: Architecture for our UNet model

UNet is a completely convolution network with residual connections between down-sampling and up-sampling layers. This allows the model to avoid problems with a similarly deep network, such a vanishing gradients, by preserving information from shallow layers in deeper layers.

We trained our UNet model similarly to our other models. We used data augmentation by randomly flipping, rotating, and cropping our images. We used the Adam optimizer with a cosine annealing learning rate scheduler with an initial learning rate of 0.0001. The smaller learning was used to improvements in the validation IoU compared to a learning rate of 0.005. Our loss function, just like withe other models, was weighted cross entropy, where the weight of each class was the multiplicative inverse of their class frequency in the test data set.

## 4 Results

The primary metric we used to evaluate the performance is the Jaccard Index, also known as the Intersection over Union (IoU). For a given category $k$, the IoU is calculated as the area of the overlap of the predicted pixels in class $k$ with the pixels of class $k$ in the ground truth divided by the area of

their union. More explicitly,

$$IoU_k = \frac{\sum_{i=1}^{I} y_k^i t_k^i}{\sum_{i=1}^{I} y_k^i + t_k^i}$$

where $y_k^i$ is 1 when the $i$-th pixel is predicted to be in class $k$ and similarly for $t_k^i$ except for the pixel belonging to the ground truth. For each class $k$, we calculated the IoU over our entire dataset and then average across our 21 classes, giving us our mean IoU. Additionally, we also used pixel accuracy as another metric, which is simply the ratio of correct predictions and total pixels.

A concise overview of our results are given in table 5.

| Model | Baseline | Improved Baseline | Custom | ResNet34 | UNet |
|---|---|---|---|---|---|
| Mean IoU | .0365 | .0177 | 0.0293 | .1288 | .0225 |
| Accuracy | 74.9% | 13.21% | 11.45% | 24.37% | 2.98% |

Table 5: Results for our 5 different models

### 4.1 Baseline

Our initial baseline model achieved a mean IoU of .0365 and an accuracy of 74.9% on the test dataset. The values of the evaluation metrics over the training epochs can be seen in 1. Additionally, a sample input, output, and ground truth mask can be seen in 2

### 4.2 Improvements over Baseline

We found that the best loss function for our model was the weighted cross entropy. While the dice and focal losses were an improvement over the initial unweighted cross entropy, neither performed as well as weighted cross entropy. In our improved training for the baseline model, we achieved an mean IoU of .0177 and an accuracy of 13.21% on the test dataset. A plot of the evaluation metrics can be seen in figure 3. Additionally a sample input, output, and target mask can be seen in figure 4.

### 4.3 Experimentation

#### 4.3.1 Custom Architecture

We managed to achieve an accuracy of 9.10% on the validation set and 10.85% on the test set at best epoch of 31, achieved by implementing an early stop. Furthermore, a mean IoU score of 0.0226 and 0.0256 was achieved on the validation and test sets respectively. Although, the pixel accuracy is a bit low from the improved baseline model, we achieve a significant improvements in terms of the mean IoU score, which highlights the segmentation accuracy.

The training and validation loss, mean IoU and accuracy plots can be visualized in Fig. 5, and the comparison between ground truth and network output mask in Fig. 6

#### 4.3.2 ResNet

On the test dataset, the pre-trained ResNet model achieved an accuracy of 24.37% and a mean IoU of .1228. The values of the various metrics over the training epochs can be seen in figure 7.

Though this is our highest IoU, the output of our model is still significantly different than the ground truth. An example input and output can be seen in the figure 8.

#### 4.3.3 UNet

Using the UNet model we achieved a test mIoU of .0225 and a test accuracy of 2.98%. The values of the evaluation metrics over the training epochs can be seen in figure 9 Additionally, a sample input, output, and mask can be seen in figure 10

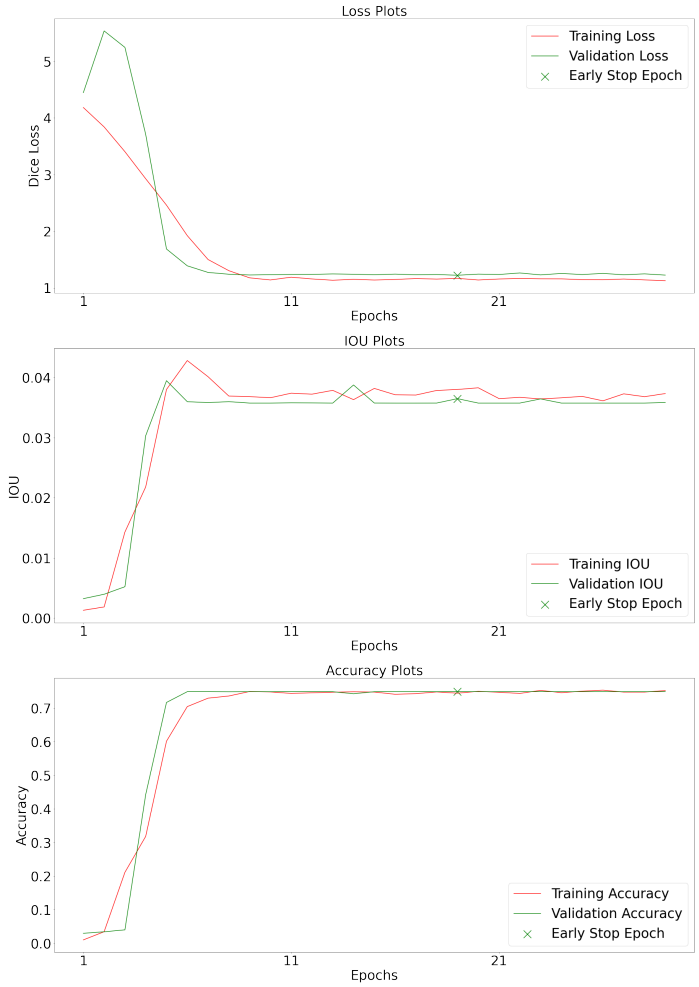


Figure 1: Loss, IoU, and Accuracy Plots for the baseline model

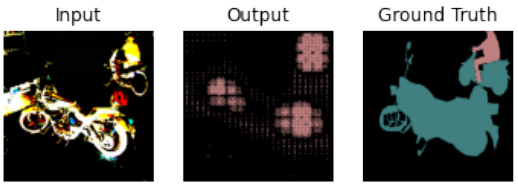


Figure 2: Sample input, output, and ground truth from the test dataset for the baseline model


Figure 3: Loss, IoU, and Accuracy Plots for the improved baseline model

# 5 Discussion

## 5.1 Baseline

Our baseline model, while achieving a high accuracy, did not really perform all that well. Though our accuracy is around 75%, most of that comes from the fact that the background class makes up the vast majority of the pixels in our training dataset. So our model only really learned that the 0 class corresponds to the background and almost every pixel is the background. We can see that it simply chooses to predict every pixel as the background as seen in figure 2. Thus, for our immediate improvements, we sought ways to rectify the class imbalance problem.

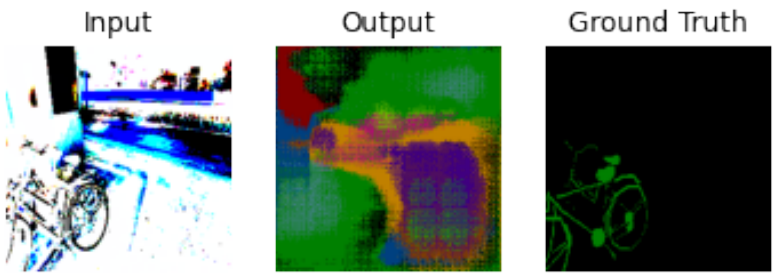


Figure 4: Sample input, output, and ground truth from the test dataset for the improved baseline model

### 5.2 Improvements over Baseline

Though the numbers on our improved baseline are worse, with both a lower accuracy and mean IoU, in some ways the performance is better. As seen in figure 4, our model no longer only predicts the background. However, now there is the inverse problem of predicting different non-background classes for what should be the background. Also, our model does seem to be finding features very well of the objects in our images, but this is likely due to the simple architecture of the model.

An additional problem we saw was overfitting, with our training loss decreasing steadily, but our validation loss plateauing almost immediately. We sought to rectify this by applying random transformations in order to augment our dataset with new images, at least from the network's perspective. While this did help slightly, the increase to mean IoU was only about 0.005.

Finally, the cosine annealling learning rate scheduler did help with convergence, allowing the model to reach a more precise minima due to smaller changes near the end of training. Similar to the data augmentation, the overall improvement was still quite minimal, with an increase in mean IoU of less than 0.01.

We also tested our model using focal and dice losses, which are also popular loss functions to use in image segmentation. However, we found that neither performed as well as the weighted cross entropy, with our without weights. These losses were originally created to deal with binary segmentation, where there is a background class and a single class of interest. However, our segmentation problem featured 20 classes of interest and this, along with our small dataset, most likely contributed to the focal loss and dice loss not working as well as they did in their original problems.

### 5.3 Experimentation

#### 5.3.1 Custom Architecture

The performance on our custom architecture is quite good w.r.t the baseline model (without improvements). It is like due to the fact that we are using skip connections, along with double-convolutions and transpose convolutions. Skip connections help us with dealing with the issue of vanishing gradients, while double convolutions allow us to capture more non-linearities/abstract relationships in the image.

We experimented with the learning rate, regularization and layer configuration. While increasing the learning rate does lead to increased accuracy sometimes, the variations in the validation and training losses are large, even in successive epochs. Regularization was fixed at $\lambda = 1e - 4$. However, the network seemed pretty resistant to the regularization change. Adding more layers helped to a certain extent, after which returns were diminished, with the epoch run-time increasing dramatically.

We see from 5 that the plot is very jittery, which can be attributed to the higher learning rate. We found that the best results were obtained using $\alpha = 5\mathrm{e}-3$. Any lower than that and the convergence rate was too low, and learning would not commence within 100 epochs. Also, interestingly, we ob-

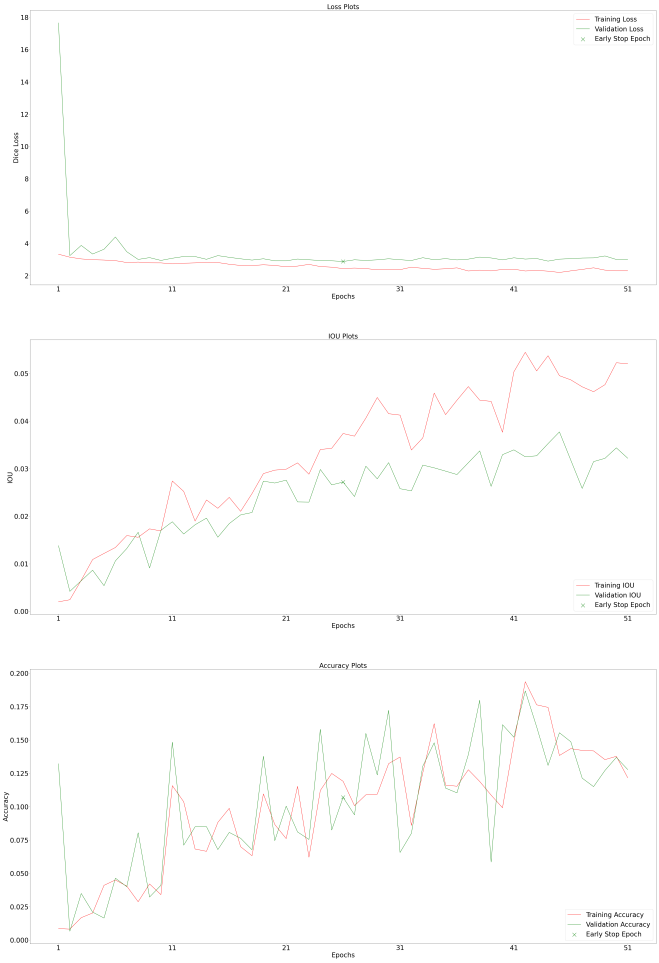


Figure 5: Loss, IoU, and Accuracy Plots for the custom FCN model

served better performance when the early stop was computed over the *mean validation IoU*, instead of the validation loss.

The segmentation results 6 are also quite good, with the network doing a decent job of segmenting the pixels (compared to the ground truth).

We see that the results are substantially worse compared to the baseline model. The reason for this is that the baseline model does not fix the class imbalance problem, and predicts most of the pixels to be background (which has many times more examples than the other labels), leading to a *false* accuracy. The segmentation results however show that our implementation does a better job in the segmentation of the input image.

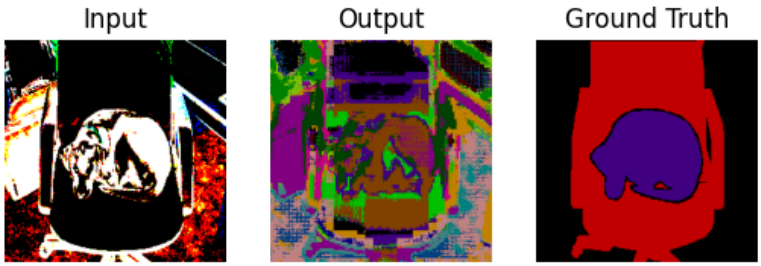


Figure 6: Sample input, output, and ground truth from the test dataset for the custom FCN model

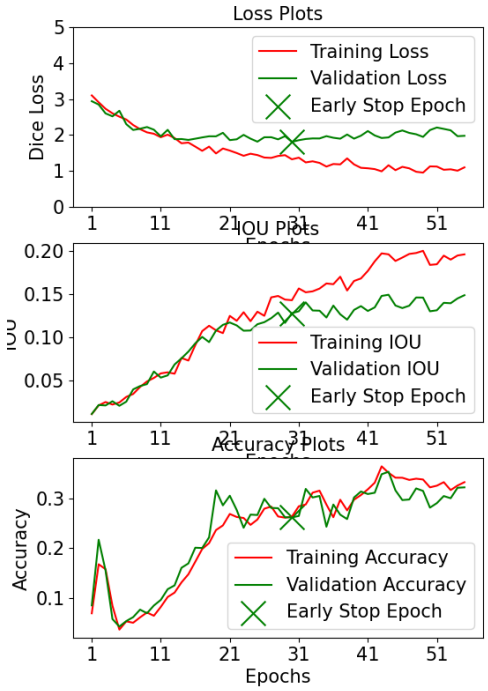


Figure 7: Loss, IoU, and Accuracy Plots for the pre-trained ResNet34 model

### 5.3.2 ResNet

Transfer learning is beneficial because it removes the time and energy needed for our model to learn the basic features that a 34-layer pre-trained model already has; hence, this is a huge time benefit since we get to reap the benefits of the optimizations that He et.al. [2] used to train their model. One drawback is the convergence time of Resnet34 versus its fewer-layered counterpart in Resnet18. Looking at the model output for Resnet34 versus Resnet18, the desired output features are slightly more recognizable in the former than the latter, although it is is still difficult to say qualitatively for sure. Quantitatively, based on the curvature of the validation loss plots, the Resnet18 converges slightly faster than the resent34. The accuracy curve of the Resnet34 network is slightly

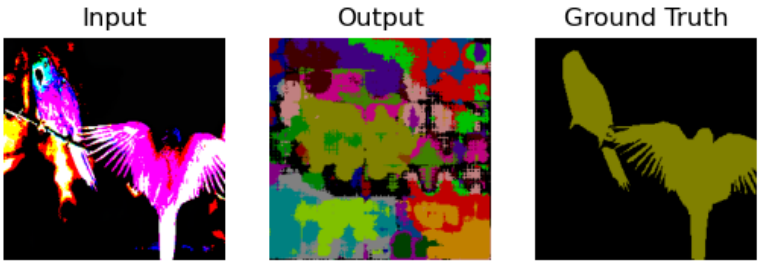


Figure 8: Sample input, output, and ground truth for a given image in the test dataset. The model used here is the pre-trained ResNet34.

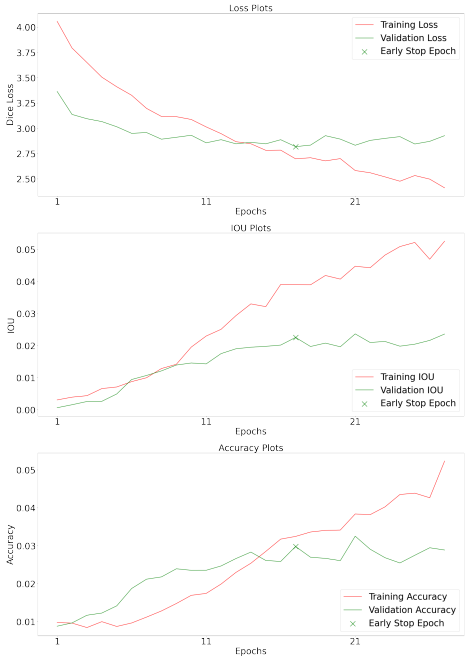


Figure 9: Loss, IoU, and Accuracy Plots for the UNet model

more volatile, it seems, than that for Resnet18, so it is possible the degradation issue returned to these layered models with our decision to freeze the model weights in order to keep the structure of Resnet34 intact as we trained our deconvolution layers and the classifier. The IOU for these networks are substantially higher than for our designed networks, likely due to the number of layers and their organization and optimization by the design team of the residual networks used. Another reason for the higher IoU score from the pre-trained ResNet is due to the fact that it was trained on thousands of images (significantly more than our 200 image training set) and thus has better generalizations to other image sets.

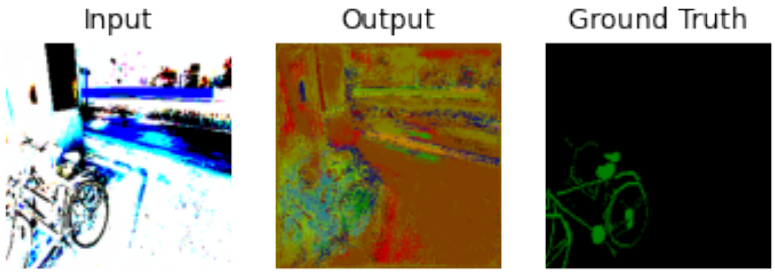


Figure 10: Sample input, output, and ground truth from the test dataset for the UNet model

#### 5.3.3 UNet

Our UNet model achieved a higher mean IoU as compared to our improved baseline model. However, we due achieve a significantly worse pixel accuracy, most likely due to the fact that the model rarely predicts pixels to be the background, which can be seen in 10. Since such a large proportion of our dataset's pixels were of the background class, this likely contributed to the rise in mean IoU and the large drop in accuracy.

Though our UNet model did not achieve very high metrics, it does seem to be better at learning features than either of our baseline models. In 10, we can see that model begins to identify the bicycle, separating it out as a different class than rest of the image, though with some noise around it. Of course it identifies the bicycle and the background mostly as wrong classes, but it has learned that the features that make up the bicycle are different than the features around it.

It is likely that the poor performance of our model is due to the limited size of our dataset. With only around 200 images to train on, our model has a difficult time being able to correctly identify all 20 classes. However, the ability for the UNet to more effectively identify features means that, of the models we trained from scratch, it would be the most likely to achieve better results on a larger dataset.

One reason for UNet being better at identifying finer features is likely due to its residual connections. These connections allow it to preserve information from earlier convolution layers later on in the network. For our baseline model, the initial convolutional layers would only receive updates via backpropagation after going through all the deconvolutional and convolutional layers above it, which likely led to vanishing gradients and thus little learning. The residual connections from UNet, on the other hand, would allow its early convolutional layers to still learn, as they would receive update from backpropagation after only backpropagating through only a few layers.

## 6 Team Reflection

### 6.1 Madeleine Kerr

I worked on setting the basic model up in order to understand the process, then we split the tasks up and I worked on Transfer learning. I figured out how to do transfer learning, implemented the Resnet34 and Resnet18 models in place of the encoder and ran experiments on those networks. I made the plots and wrote the sections related to that part of the project.

### 6.2 Cameron Cinel

I got the initial fully convolutional network and training/testing loop up and running. I also implemented the additional loss functions and other baseline model improvements. In addition, I implemented the UNet model and created the helper functions to plot loss, IoU, and accuracy functions as well as output images. As far as writing, I wrote the sections on the baseline, baseline improvements, and UNet from both the methods, results, and discussion.

### 6.3 Rishabh Bhattacharya

I focused on experimenting with alternate image transforms, and setting up the initial skip connection network, and later incorporated it into the FCN8 implementation. For the report, I contributed the introduction, discussion of custom architecture, and generated multiple plots and ran tests to tabulate the accuracy.

### 6.4 Shrey Kansal

I focused on experimenting with the implementation of custom architecture. I read papers and online blogs related to implementing FCN in pytorch for semantic segmentation. I made the plots, output semantic features and wrote the sections related to that part of the project. For the report, I contributed to Abstract and Related Work sections as well.

## References


[1] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[4] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[5] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[6] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2016.

[7] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571, 2016.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[9] Justin Sirignano and Konstantinos Spiliopoulos. Scaling limit of neural networks with the xavier initialization and convergence to a global minimum, 2019.